Optimal Solutions for a Geometric Knapsack Problem using Integer Programming*

Rafael G. Cano† Cid C. de Souza† Pedro J. de Rezende†

Abstract

The objective of this paper is to present an experimental study of the Geometric Knapsack Problem (GKP) with the goal of obtaining provably optimal solutions. We introduce an Integer Linear Programming model for the GKP and apply it to hundreds of instances of two classes: one comprised of uniformly generated points with randomly assigned values; and another composed of convex layered points with value distribution biased towards concentrating negative-valued points on the innermost layers. Trial tests were used to guide the choice of input parameters so as to avoid generating trivial instances. Our experiments show that the layered class is significantly harder to be solved to optimality, in practice, since even instances with as few as 35 points could not be solved within 5 minutes of CPU time.

1 Introduction

Geometric knapsack problems (GKP) are extensions of the classic knapsack problem to a geometric setting. In the classic version, we are given a set of items with specified *weights* and *values*, together with a knapsack of limited capacity. The objective is to select a subset of items whose combined weight does not exceed the capacity of the knapsack and whose total value is maximum.

The geometric variants typically consist of the so-called *fence enclosure problems* [1]. Here, we restrict our study to a two-dimensional version in which items are points in the plane. Consider a set $P = \{p_1, p_2, \ldots, p_n\}$ of n distinct points. With each point $p_i \in P$ there is an associated real value v_i, unrestricted in sign. The "knapsack" (also called *fence*) consists of a simple polygon, and the selected items are the points that it encloses. Unlike the classic variant, items do not have an explicit weight. However, there is a cost associated with the total length of the fence. The objective is to maximize the *net profit* given by the total value of the enclosed points minus the cost of the fence. An example is shown in Figure 1.

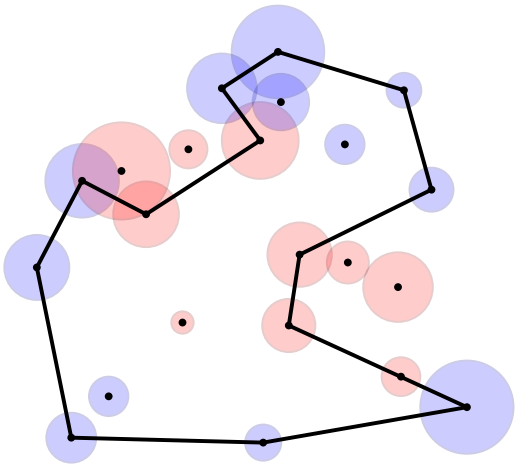

Figure 1: An instance of the GKP with an optimal fence. Positive and negative point values are represented by blue and red circles, resp., with radii proportional to the magnitude of the values.

Formally, we say that a simple polygon φ (a fence) *strictly encloses* all the points in P that lie in the interior of the region bounded by φ. On the other hand, among the points that lie *on* the polygon boundary, φ also *encloses* those of non-negative value. The reason for this apparent asymmetry is that, if we only take into account the points that are strictly enclosed by the fence, then an exact solution, such as the one shown in Figure 1, would not be attainable. Nonetheless, our extended definition of enclosure allows us to compute a polygon that realizes the supremum of the net profit function. Moreover, it is always possible to slightly alter the given polygon in such a way that it strictly encloses all positive-valued points lying on its boundary. This modification would increase the length of the fence by some $\varepsilon > 0$, but the resulting additional cost could be made as small as desired.

We denote by P_φ the set of all points enclosed by φ. Let L_φ be the total Euclidean perimeter of φ, and $c \geq 0$ be a construction cost per unit of length of the fence. The GKP requires a fence φ to be built, which maximizes the net profit $\left(\sum_{p_i \in P_\varphi} v_i\right) - c \cdot L_\varphi$.

*This work was supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) #304727/2014-8, #309627/2017-6, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) #2014/12236-1, #2018/11100-0, and Fundo de Apoio ao Ensino, à Pesquisa e Extensão (FAEPEX).

†Institute of Computing, University of Campinas, Campinas – Brazil, {rgcano,cid,rezende}@ic.unicamp.br

1.1 Related Work

The GKP was proposed by Arkin et al. [1], who introduced its many variants. In their work, they allow the set of items to consist of either points, line segments or simple polygons. They also consider versions in which there is an upper bound on the total length of fence available. For point items, they show that the problem is NP-hard if values are unrestricted in sign (which is precisely the variant that we address here). However, if all values are non-negative, they describe two exact $O(n^3)$ algorithms. Moreover, if there is limited length of fence to use, the problem of maximizing the total value of the enclosed points is NP-hard even with non-negative values. For polygonal items, they present different polynomial-time algorithms, depending on whether the fence is allowed to cross the objects.

Our version of the GKP is also related to the following problem. Given two sets of points R (the "red points") and B (the "blue points"), we say that a *separating polygon* for R and B is a simple polygon φ such that all points of R are in the interior or on the boundary of φ, and all points of B are in the exterior or on the boundary of φ (or vice-versa). The Red-Blue Separation Problem (RBSP) consists of finding the minimum-perimeter separating polygon for the given sets of points. This problem was shown to be NP-hard by Eades and Rappaport [5]. Approximation algorithms were proposed by Mata and Mitchell [6] and Arora and Chang [2].

Reinbacher et al. [8] study an enclosure problem in the context of Geographic Information Systems (GIS). They address the problem of computing boundaries to imprecise, verbally-defined regions, such as "Northern Portugal" or "British Midlands". In order to estimate the boundary, they first extract a set of known locations that are generally considered to be inside or outside the desired region. Then, they compute a separating polygon that encloses the inside points and obeys certain geographic criteria.

Finally, our work also benefits from the theory developed for the Traveling Salesman Problem (TSP). In particular, we use some results presented by Balas [3] for the Prize Collecting TSP. In this version, a salesman travels between pairs of cities and collects an amount of prize money at each city that he visits. Contrary to the classic variant, he can choose not to visit a city at the cost of a penalty. The goal is to visit enough cities so as to collect a minimum specified amount of prize money while minimizing travel and penalty costs.

1.2 Our Contribution

In this work, we present a formulation of the GKP as an Integer Linear Program (ILP). This is, to the best of our knowledge, the first exact algorithm for this problem. We use the ILP to obtain provably optimal solutions for instances with up to 40 points. Based on a series of experiments, we also devised a class of instances that are particularly challenging to be solved in practice.

The remainder of the text is organized as follows. Section 2 discusses some properties of optimal solutions. Section 3 presents our ILP. Section 4 describes the instances used in our tests and reports the main results of our experimental evaluation. Some final remarks are provided in Section 5.

2 Structure of Optimal Solutions

Given an instance of the GKP, consider an optimal fence $\varphi*$ that encloses the set of points $P_{\varphi*}$. Clearly, $\varphi*$ must be the minimum-perimeter polygon that separates $P_{\varphi*}$ and $P \setminus P_{\varphi*}$. Thus, properties that apply to optimal solutions of the RBSP also apply to the GKP. In particular, for both problems, all vertices of the constructed polygon must be points of the given set. This is illustrated in Figure 2, where points of P are shown as solid disks. Since vertices r and s are not points of P, a polygon of smaller perimeter can be obtained using the chords $\overline{r'r''}$ and $\overline{s's''}$, for the triangles (r', r, r'') and (s', s'', s) do not contain any points of P.

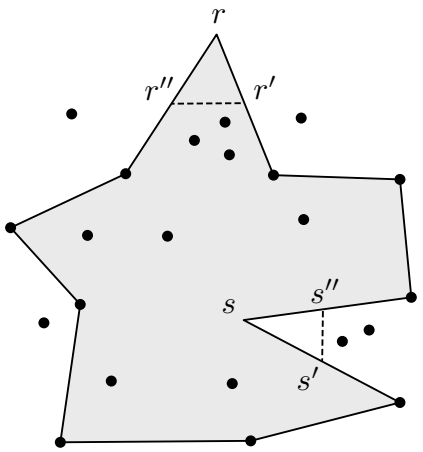

Figure 2: A suboptimal fence with two vertices r and s that are not points of P.

Moreover, it is easy to obtain a stronger result for the GKP. Suppose a convex vertex r is a point of P with negative value. By definition, r is not enclosed by the fence. Thus, the previous perimeter-reducing construction would still maintain the same set of enclosed points and the original fence would not be optimal. Similarly, if $s \in P$ is a reflex vertex of positive value, by the aforementioned construction, a fence of smaller perimeter could be built. Therefore, all convex (reflex) vertices of an optimal fence must be points of P of positive (negative) values.

3 Integer Programming Model

Let $G = (P, A)$ be the complete directed graph whose node set is the set of input points P and $A = \{(p_i, p_j) : p_i, p_j \in P, i \neq j\}$. Given three points $p_i, p_j, p_k \in P$, let $\Delta(p_i, p_j, p_k)$ denote their orientation, i.e., the sign of the signed area of the triangle p_i, p_j, p_k. To simplify our exposition, we assume that the points of P are in general position. Hence, $\Delta(p_i, p_j, p_k) \neq 0$ for any three points $p_i, p_j, p_k \in P$.

In our formulation, we represent the fence as a single directed cycle in G. For each arc $(p_i, p_j) \in A$ we define a binary variable x_{ij} that takes value 1 if and only if this arc is part of the fence. Also, for each point p_i we define a binary variable y_i that indicates whether p_i is enclosed by the fence. Two auxiliary expressions will be used to simplify the description of the constraints. First, note that a point p_i is a vertex of the fence whenever there is an arc incident to p_i. Thus, we define

$$B(p_i) = \sum_{(p_i,p_j)\in A} x_{ij}. \tag{1}$$

Since the solution must consist of a single directed cycle, we include a set of constraints that limit the number of outgoing and incoming arcs on each point to at most one. Thus, $B(p_i)$ may take one of two values: 1 if p_i is on the fence, and 0 otherwise.

In order to model the objective function, it is also necessary to determine which points are enclosed by a fence. Ultimately, we must solve the point location problem w.r.t. an arbitrary simple polygon built from (a subset of) the input points. We apply an approach similar to the following parity checking ray-crossing algorithm. Denote by ℓ_i the horizontal half-line that extends from a point p_i to the positive direction of the x-axis. Given a simple polygon φ, p_i is interior to φ iff ℓ_i crosses (the boundary of) φ an odd number of times.

However, parity checking in a linear model is not a simple task. Nonetheless, arc directions are useful to circumvent this issue. First, we adopt the convention that the cycle representing the fence must be counterclockwise-oriented (we will later show how to enforce this in the model). Now, given a point p_i not on the border of the fence, we trace ℓ_i starting at p_i, as shown in Figure 3. Each time we cross an arc (p_j, p_k), we inspect the value of $\Delta(p_i, p_j, p_k)$. If it is positive, we are moving from the inside to the outside of the fence, otherwise we are moving from the outside to the inside. Let n_i^{out} and n_i^{in} be the number of times we move to the outside and to the inside of the fence, respectively. Clearly, p_i is outside the fence iff $n_i^{\text{out}} = n_i^{\text{in}}$.

Let $R_i \subseteq A$ be the set of arcs crossed by ℓ_i (excluding the intersections at p_i itself, if any). We further define $R_i^+ = \{(p_j, p_k) \in R_i : \Delta(p_i, p_j, p_k) > 0\}$ and $R_i^- = \{(p_j, p_k) \in R_i : \Delta(p_i, p_j, p_k) < 0\}$. From the previous observation, we may write

$$n_i^{\text{out}} = \sum_{(p_j,p_k)\in R_i^+} x_{jk} \qquad \text{and} \qquad n_i^{\text{in}} = \sum_{(p_j,p_k)\in R_i^-} x_{jk}.$$

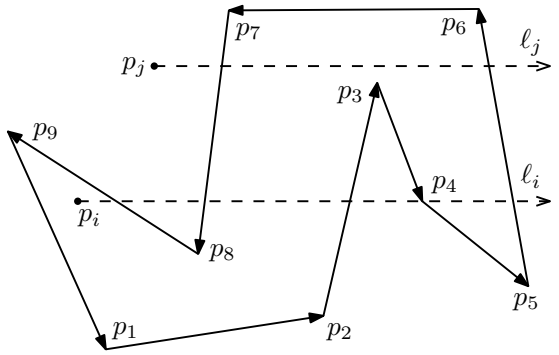

Figure 3: Illustration of the ray-crossing algorithm used to determine which points are enclosed by the fence.

As an example, for the point p_i in Figure 3, all variables in the first summation have value 0, except for $x_{8,9}$, $x_{2,3}$ and $x_{5,6}$, so $n_i^{\text{out}} = 3$. In the second summation, the only variables with value 1 are $x_{7,8}$ and $x_{3,4}$, so $n_i^{\text{in}} = 2$. Note that ℓ_i intersects both (p_3, p_4) and (p_4, p_5) at p_4. Whenever the half-line intersects the fence at a vertex p_k, we only consider that it crosses those arcs whose other endpoint is above p_k. Thus $x_{4,5}$ is not included in the summation. Finally, we define

$$I(p_i) = n_i^{\text{out}} - n_i^{\text{in}}. \tag{2}$$

From the previous discussion, it follows that for any point p_i not on the border of the fence, $I(p_i)$ has value 1 when p_i is inside the fence and 0 otherwise. It should be noted that these values may differ for points lying on the fence (e.g., in Figure 3, $I(p_3) = 1$ and $I(p_5) = 0$). Even in this case $I(p_i)$ can only take values 0 or 1. As we argue in the next section, these cases will not be of importance for our formulation.

It remains for us to show how to enforce a counterclockwise orientation on the constructed cycle. In order to do this, we make use of the observations in Section 2. Since positive-valued points cannot be reflex vertices of an optimal fence, we write constraints to enforce this property. Let $p_i, p_j, p_k \in P$ be three distinct points, with $v_j > 0$ (recall that v_j denotes the value associated with point p_j). If $\Delta(p_i, p_j, p_k) < 0$, then p_j becomes a reflex vertex whenever arcs (p_i, p_j) and (p_j, p_k) are used together in a fence; thus we may write $x_{ij} + x_{jk} \leq 1$. Analogously, if $v_j < 0$ and $\Delta(p_i, p_j, p_k) > 0$, the previous inequality also prevents p_j from becoming a convex vertex. In general, this inequality is valid whenever $v_j \cdot \Delta(p_i, p_j, p_k) < 0$ and it guarantees that all cycles will lead to counterclockwise-oriented polygons.

Given an arc $(p_i, p_j) \in A$, let d_{ij} denote the Euclidean distance between p_i and p_j. Also, given a set of points

$S \subset P$, let $\delta(S)$ denote the set of all arcs of G directed from a point in S to a point in $P \setminus S$, i.e., $\delta(S) = \{(p_i, p_j) \in A : p_i \in S \text{ and } p_j \in P \setminus S\}$.

The following ILP is a formulation of the GKP. The objective function to be maximized is

$$\sum_{p_i \in P} v_i \cdot y_i - \sum_{(p_i,p_j) \in A} c \cdot d_{ij} \cdot x_{ij} \tag{3}$$

subject to the following constraints:

$$\sum_{(p_i,p_j) \in A} x_{ij} \leq 1 \qquad \forall\, p_i \in P \tag{4}$$

$$\sum_{(p_i,p_j) \in A} x_{ij} = \sum_{(p_j,p_i) \in A} x_{ji} \qquad \forall\, p_i \in P \tag{5}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall\, p_i, p_j \in P \tag{6}$$

$$x_{ij} + x_{jk} \leq 1 \qquad \forall\, p_i, p_j, p_k \in P : \; v_j \cdot \Delta(p_i, p_j, p_k) < 0 \tag{7}$$

$$y_i \leq I(p_i) + B(p_i) \qquad \forall\, p_i \in P : v_i > 0 \tag{8}$$

$$y_i \geq I(p_i) - B(p_i) \qquad \forall\, p_i \in P : v_i < 0 \tag{9}$$

$$\sum_{(p_i,p_j) \in \delta(S)} x_{ij} \geq B(p_k) + B(p_\ell) - 1 \qquad \forall\, S \subset P,\; 2 \leq |S| \leq n-2,\; p_k \in S, p_\ell \in P \setminus S. \tag{10}$$

The objective function (3) computes the sum of the values of all enclosed points minus the sum of the costs of all arcs in the cycle that represents the fence. Constraints (4) limit the number of outgoing arcs of each node to at most one, and constraints (5) ensure that whenever an arc enters a node, another arc must leave it. Constraints (6) prevent two opposite arcs from being chosen. Constraints (7) guarantee that positive-valued points cannot become reflex vertices of the fence, and negative-valued points cannot become convex vertices.

Constraints (8) and (9) enforce the desired meaning of the y variables. If p_i has positive value, then, due to the maximization of the objective function, y_i will be set to 1, except if some constraint forbids it. Hence, we must only force it to be 0 when p_i is strictly outside the fence. This is done by constraints (8). Similarly, if p_i has negative value, y_i will be set to 0 since we are maximizing (3), unless it violates some constraint. Thus, we must only force it to be 1 when p_i is strictly inside the fence. This is accomplished by constraints (9).

Finally, we must guarantee that a single cycle will be constructed by the model. This is done by constraints (10). These constraints are well-known in the literature of the TSP. The classic TSP requires that a cycle be constructed using all nodes of the input graph (i.e., it must be a Hamiltonian cycle). However, in our case, points can be left out, so we use inequalities that were originally studied by Balas [3] for the Prize Collecting TSP. Those inequalities state that, given a subset S of the input points, if points $p_k \in S$ and $p_\ell \in P \setminus S$ are nodes on a constructed cycle, then there must be at least one arc from S to $P \setminus S$. Although there is an exponential number of them, we used a well-known procedure to separate them in polynomial time using a max-flow min-cut algorithm (see, e.g., Padberg and Rinaldi [7]).

As a final remark, we mentioned earlier that $I(p_i)$ might take value 0 or 1 if p_i is on the fence itself. Note that this variation does not affect the correctness of our model, since constraints (8) and (9) are trivially satisfied when $B(p_i) > 0$.

4 Experiments

We now present our experimental study of the GKP, which had two main goals. Firstly, we wanted to understand what features make an instance particularly hard to solve in practice. This led us to create a challenging set of benchmark instances for exact methods. Secondly, we wished to evaluate the performance of the proposed ILP and to determine which instance sizes can be solved in a reasonable amount of time. More details are given in the next sections.

4.1 Instances

In order to create instances for the GKP, we must decide on two major factors: the spatial distribution of the points and the assignment of their associated values. In this work, we limit ourselves to uniformly distributed points in a square. The magnitudes of point values are also drawn from a uniform distribution and are restricted to a predefined range. We do, however, vary the way in which the sign of each value is chosen, giving rise to two classes of instances.

In the first class, each value is given an equal probability of receiving a positive or a negative sign. We refer to these instances as *uniform instances*. Uniform instances exhibit no particular structure and are useful as a baseline to evaluate the performance of our ILP.

For the second class, we attempt to create more challenging instances based on the following idea. Since the objective function seeks to maximize the net profit, an optimal fence will naturally enclose as many positive-valued points as possible, while avoiding the enclosure of negative ones. Thus, an instance becomes more difficult when negative-valued points are hard to avoid. So, we assign positive values to the outermost points, and gradually increase the probability that a point receives a negative value as we move inwards through the point set.

More formally, to create instances for the second class, we start by computing the convex layers of the set of points. Given a point p_i, denote by d the depth of its layer and by D the depth of the innermost layer. The

probability that p_i receives a negative value is set to $\sqrt{d/D}$ (we consider that the depth of the first layer – the convex hull – is zero). Here, we apply the square root so as to obtain a higher number of negative-valued points, otherwise, the instance loses some of its complexity, as we determined through some preliminary experiments. We refer to instances of this class as *layered instances*.

We conclude this section by providing further details on the creation of our instances. The square from which the n points are drawn has side $100\sqrt{n}$. The magnitude of the values is selected in the range $[50, 150]$. The lower bound of 50 is used because values with very small magnitudes rarely have any significant impact on optimal solutions, so the associated points often become irrelevant. The complete set of instances can be found on our web page [4].

4.2 Computational Results

We executed the experiments on an Intel Xeon E5-2603 1.60GHz CPU with 32GB of RAM. Integer programs were solved with a branch and cut algorithm using CPLEX 12.8 in deterministic mode using a single thread. Our code was written in C++ and compiled with g++ 5.4.0 with optimization flag `-O3`.

Initially, we ran some preliminary experiments to observe the behavior of each class of instances with different values of the cost c per unit length of fence. Note that if c is too high, the optimal fence will enclose a single point of maximum value, and the instance becomes trivial. Similarly, if c is too low, the cost of the fence becomes negligible. In this case, the optimal fence will enclose all (and only) positive-valued points, thus, the problem reduces to the RBSP. To avoid both scenarios, we chose intermediate values for c. For each class, we ran tests with three values that were found to be the most suitable according to the results of our preliminary experiments: $\{0.4, 0.6, 0.8\}$ and $\{0.3, 0.4, 0.5\}$ for uniform and layered instances, respectively.

We created instances of five different sizes: $\{20, 25, 30, 35, 40\}$. For each size, we created 10 instances of each class. Thus, in total, we have 50 uniform and 50 layered instances. We, then, ran our ILP on each one of them with the three specified cost values, yielding a total of 300 test problems. The results are summarized in Tables 1 and 2. For each size n and cost c, we report the minimum (tmin) and maximum (tmax) running times of the 10 instances. We also show the average running time (tavg) together with the standard deviation. We imposed a time limit of 5 minutes for each run, and the last column shows the number of instances solved to optimality within this time bound.

The results for uniform instances (Table 1) indicate that this class does not pose difficulties to our ILP, which was able to find optimal solutions for all instances. Although the average running time is always below one minute, there is a very large deviation, which is, in most cases, larger than the average itself. In several cases, the solver took a long time to find a good feasible solution to the ILP, which led to the processing of a high number of nodes in the branch and bound tree and, consequently, high running times.

c	n	tmin	tmax	tavg	#opt
0.4	20	0.1	0.6	0.3 ± 0.2	10
	25	0.2	11.8	2.2 ± 3.6	10
	30	0.2	16.6	5.4 ± 6.2	10
	35	0.5	57.7	10.8 ± 17.6	10
	40	1.6	201.5	52.6 ± 68.8	10
0.6	20	0.1	1.7	0.3 ± 0.5	10
	25	0.1	6.0	1.0 ± 1.8	10
	30	0.2	10.2	2.7 ± 3.1	10
	35	0.4	43.9	10.0 ± 13.5	10
	40	1.4	157.3	39.3 ± 47.9	10
0.8	20	0.1	0.3	0.1 ± 0.1	10
	25	0.1	1.2	0.3 ± 0.3	10
	30	0.1	1.9	0.8 ± 0.6	10
	35	0.4	54.3	11.0 ± 16.4	10
	40	1.0	178.9	50.7 ± 62.5	10

Table 1: Summary of the results for uniform instances. Times are given in seconds.

c	n	tmin	tmax	tavg	#opt
0.3	20	0.2	1.2	0.6 ± 0.4	10
	25	0.3	52.4	13.5 ± 16.9	10
	30	6.3	231.7	91.5 ± 72.6	10
	35	0.6	300.0	200.0 ± 128.7	5
	40	59.4	300.0	251.0 ± 89.9	3
0.4	20	0.2	3.0	0.9 ± 0.9	10
	25	0.4	63.0	16.0 ± 21.8	10
	30	3.1	120.6	34.4 ± 39.2	10
	35	0.7	300.0	153.1 ± 116.7	8
	40	14.9	300.0	235.0 ± 106.3	4
0.5	20	0.1	2.4	1.1 ± 0.9	10
	25	0.3	38.3	11.0 ± 12.8	10
	30	0.3	237.2	38.4 ± 72.5	10
	35	1.2	300.0	119.4 ± 126.1	9
	40	2.5	300.0	204.9 ± 128.1	4

Table 2: Summary of the results for layered instances. Times are given in seconds.

As for layered instances, the results in Table 2 show that this is, as desired, a much harder class. We were not able to find optimal solutions for several instances with 35 and 40 points. The average running times are also higher in all cases. Therefore, we believe this is a challenging set of instances to serve as a benchmark for the GKP.

In our last experiment, we examined the behavior of individual instances for a wide range of fence cost values.

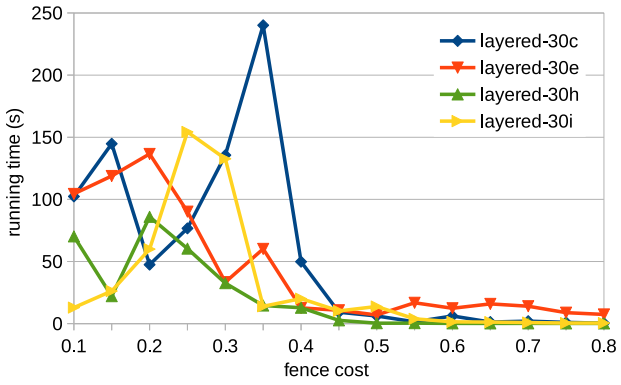

Figure 4: Running time for each value of c for four layered instances with 30 points each.

We selected four layered instances and ran our ILP on them with values of c that range from 0.10 to 0.80 in increments of 0.05. The running times are depicted in Figure 4.

The graph shows that for low values of c, the instances are harder to solve. As we mentioned earlier, when c is too low, the problem reduces to the RBSP, which is also NP-hard. The high variance in the running times is due to CPLEX's branch and bound process, which is, to a certain extent, unpredictable. If the solver cannot find a good solution quickly, a lot of branching is required and the computation time increases. However, for high values of c, the problem becomes easier. This happens because the cost of several edges becomes prohibitive and the solver can easily prune some nodes of the branch and bound tree.

We observed this behavior for all test instances. Although the specific value of c varies among instances, there is always some threshold for which they become trivial. This is illustrated in Figure 5, which shows four optimal solutions for the instance layered-30c, obtained with four different values of c. The rightmost figure shows the shortest optimal non-degenerate fence for that instance. If c is further increased, the optimal solution becomes a degenerate polygon enclosing a single point.

5 Conclusion

We addressed an NP-hard variant of the Geometric Knapsack Problem and proposed an ILP formulation for it. This is, to the best of our knowledge, the first exact method to solve this variant. We also devised two classes of instances to evaluate our ILP. One of them, namely, layered instances, proved to be quite challenging for our formulation, and we propose it as a benchmark for this problem. As for future work, we believe the development of effective heuristics could improve the running times of our ILP, since in several cases, CPLEX struggled to find feasible solutions. Finally, the study of known optimal solutions could also lead to significant speedups, especially if one can develop preprocessing procedures that take advantage of the geometric properties of each instance.

References

[1] E. M. Arkin, S. Khuller, and J. S. Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993.

[2] S. Arora and K. Chang. Approximation schemes for degree-restricted MST and red–blue separation problems. *Algorithmica*, 40(3):189–210, 2004.

[3] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[4] R. G. Cano, C. C. de Souza, and P. J. de Rezende. Geometric knapsack – instances and experimental results, 2018. `www.ic.unicamp.br/~cid/Problem-instances/Geometric-Knapsack`.

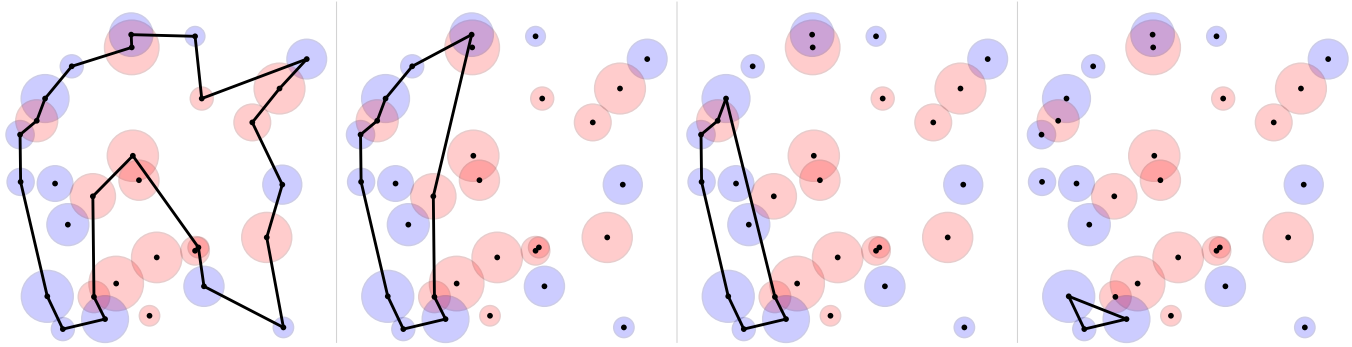

Figure 5: Optimal fences for the instance layered-30c with fence costs of 0.25, 0.40, 0.60 and 0.75, from left to right. Positive and negative point values are represented by blue and red circles, resp., with radii proportional to the magnitude of the values.

[5] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14(9):715–718, 1993.

[6] C. S. Mata and J. S. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 360–369. ACM, 1995.

[7] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47(1):219–257, 1990.

[8] I. Reinbacher, M. Benkert, M. van Kreveld, J. S. Mitchell, J. Snoeyink, and A. Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, 50(3):386–414, 2008.